Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КУРСОВАЯ РАБОТА**

**ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «Доставка еды (на двоих)»**

**ПМ.05 «ПРОЕКТИРОВАНИЕ И разработкА ИНФОРМАЦИОННЫХ СИСТЕМ»**

**МДК 05.03 «ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»**

**09.02.07 «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | Черный Д.С. |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |

Тольятти, 2022

Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

|  |  |  |
| --- | --- | --- |
|  |  | Утверждаю:  Заместитель директора по УР  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.С. Киронова  *« » 2022 г.* |

**ЗАДАНИЕ**

на курсовую работу

**по ПМ.05 Проектирование и разработка информационных систем модуля, выполняемой в рамках МДК.05.03 Тестирование информационных систем**

студента группы ИСП-32

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Черного Данила Сергеевича \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия Имя отчество студента*

|  |  |
| --- | --- |
| Тема курсовой работы : | «Доставка еды (на двоих)» |

1. Содержание задания:

1.1 Тестирование информационной системы.

1. Исходные данные:

Исходные данные для практической реализации автоматизированной информационной системы (АИС) берутся из различных информационных источников (Интернет-ресурсы, печатные издания, периодика и др.).

1. Содержание курсовой работы

Введение

1 Анализ методов тестирования

1.1 Критерии и принципы тестирования

1.2 Методы тестирования

2 Тестирование информационной системы (название)

2.1. Разработка тестовой документации (тест-дизайн)

2.2. Разработка тестовых сценариев

Заключение

Список использованных источников

Приложения

Дата выдачи задания: « 17 » января 2022г.

Дата сдачи работы на отделение: « » 2022г.

Руководитель курсового(ой) проекта(работы) \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись расшифровка подписи

Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КАЛЕНДАРНЫЙ ПЛАН**

выполнения курсовой работы

Студентом 3 курса ИСП-32 группы Черный Данила Сергеевич

*Фамилия, И.О.*

По теме **Учёт контингента учащихся учебного заведения**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  этапа  работы | Содержание этапов работы | Плановый срок выполнения этапа | Планируемый объем выполнения  этапа, % | Отметка  о  выполнении  этапа |
| 1 | Выбор, обоснование темы и объекта исследования | Январь 2022 | 5% |  |
| 2 | Утверждение темы, согласование плана. Введение, библиография | Январь 2022 | 10% |  |
| 3 | Изучение и анализ информационных материалов по теме | Февраль 2022 | 15% |  |
| 4 | Обоснование актуальности выбранной темы применительно к профессиональной деятельности (введение) | Февраль 2022 | 20% |  |
| 5 | Изложение материала основной части по теме курсовой работы | Февраль 2022 | 20% |  |
| 6 | Подведение итогов проведенного анализа, формулировка выводов УИР применительно к профессиональной деятельности (заключение) | Март 2022 | 20% |  |
| 7 | Оформление работы и сдача на проверку | Март 2022 | 10% |  |
| 8 | Защита работы |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | Черный Д.С. |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc100908991)

[1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ 7](#_Toc100908992)

[1.1 Критерии и принципы тестирования 7](#_Toc100908993)

[1.2 Методы тестирования 8](#_Toc100908994)

[2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ (НАЗВАНИЕ) 11](#_Toc100908995)

[2.1. Разработка тестовой документации (тест-дизайн) 11](#_Toc100908996)

[2.2. Разработка тестовых сценариев 14](#_Toc100908997)

[ЗАКЛЮЧЕНИЕ 20](#_Toc100908998)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc100908999)

# ВВЕДЕНИЕ

В проектах по разработке программного обеспечения (ПО) помимо основной задачи по реализации заявленной функциональности существует не менее важная задача по обеспечению качества ПО. Качество ПО (Software quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности. Каждый участник проекта может иметь собственное представление о критериях качества и оценке степени присутствия критериев качества в ПО. Таким образом, возникает задача определения круга заинтересованных лиц, согласования набора критериев качества и нахождения оптимального баланса критериев качества, обеспечивающего качественное ПО. Одним из устоявшихся способов контроля качества является тестирование. Тестирование ПО (Software testing) — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

Автоматизированная информационная система данных осуществляет следующие работы:

* хранение, редактирование;
* ведение, созданием диаграмм и отчетов.

Актуальность темы: определяется тем, что в связи с моральным и физическим устареванием платформы, форматов невозможно использовать существующую автоматизированную информационную систему, которая замедляет работу, частыми техническими сбоями в системе и определяется актуальность разработки информационной системы «Доставка еды (на двоих)» предприятия.

Выше изложенное в целом на теоретико-методологическом уровне определило проблему настоящего исследования: разработка программного модуля «Доставка еды (на двоих)»

1. Анализ предметной области.
2. Разработка функциональной модели.
3. Разработка логической и физической структуры.
4. Разработка информационной системы «Доставка еды (на двоих)».
5. Разработка интерфейса ИС.
6. Разработка руководства пользователя.
7. Тестирование разработанного ПП.

Методы исследования: Для разработки информационной системы будут применяться методы анализ деятельности предприятия с выявлением его функции, а также функциональное моделирование системы.

# 1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ

## 1.1 Критерии и принципы тестирования

Структурные критерии используют информацию о структуре программы, что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурные критерии базируются на основных элементах графа управления - операторах, ветвях и путях.

* Условие критерия тестирования команд (критерий С0) - набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза.
* Условие критерия тестирования ветвей (критерий С1) - набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза.
* Условие критерия тестирования путей (критерий С2) - набор тестов в совокупности должен обеспечить прохождение каждого пути не менее 1 раз.

Принцип 1 — Тестирование демонстрирует наличие дефектов (Testing shows presence of defects). Тестирование может показать, что дефекты присутствуют, но не может доказать, что их нет. Тестирование снижает вероятность наличия дефектов, находящихся в программном обеспечении, но не гарантирует их отсутствия.

Принцип 2 — Исчерпывающее тестирование недостижимо (Exhaustive testing is impossible). Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев. Вместо исчерпывающего тестирования должны использоваться анализ рисков и расстановка приоритетов, чтобы более точно сфокусировать усилия по тестированию.

Принцип 3 — Раннее тестирование (Early testing). Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки ПО или системы, и должны быть сфокусированы на определенных целях.

Принцип 4 — Скопление дефектов (Defects clustering). Разные модули системы могут содержать разное количество дефектов – то есть плотность скопления дефектов в разных частях кода может отличаться. Усилия по тестированию должны распределяться пропорционально фактической плотности дефектов. В основном, большую часть критических дефектов находят в ограниченном количестве модулей.

Принцип 5 — Парадокс пестицида (Pesticide paradox). Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов. Чтобы преодолеть этот «парадокс пестицида», тестовые сценарии должны регулярно рецензироваться и корректироваться, новые тесты должны быть разносторонними, чтобы охватить все компоненты программного обеспечения, или системы, и найти как можно больше дефектов.

Принцип 6 — Тестирование зависит от контекста (Testing is concept depending). Тестирование выполняется по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем новостной портал.

Принцип 7 — Заблуждение об отсутствии ошибок (Absence-of-errors fallacy). Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Система должна быть удобна пользователю в использовании и удовлетворять его ожиданиям и потребностям.

## 1.2 Методы тестирования

## Первый метод - тестирование Black-Box

Методика тестирования без каких-либо знаний о внутренней работе приложения называется «черным ящиком». Тестер не обращает внимания на архитектуру системы и не имеет доступа к исходному коду. Как правило, при выполнении теста с «черным ящиком» тестер будет взаимодействовать с пользовательским интерфейсом системы, предоставляя входные данные и анализируя выходы, не зная, как и где обрабатываются входы.

Общие шаги, которые необходимо выполнить для проведения любого типа тестирования черного ящика.

* Сначала рассматриваются требования и спецификации системы.
* Тестер выбирает допустимые входные данные (положительный сценарий тестирования), чтобы проверить, правильно ли их обрабатывает SUT. Кроме того, некоторые недействительные входные данные (сценарий отрицательного тестирования) выбираются для проверки того, что SUT может их обнаружить.
* Тестер определяет ожидаемые результаты для всех этих входов.
* Тестировщик программного обеспечения создает тестовые наборы с выбранными входами.
* Тестовые случаи выполнены.
* Тестер программного обеспечения сравнивает фактические результаты с ожидаемыми результатами.
* Дефекты, если таковые имеются, исправлены и перепроверены.

## Типы тестирования черного ящика

Существует много видов тестирования черного ящика, но наиболее важными являются следующие:

* **Функциональное тестирование** — этот тип тестирования черного ящика связан с функциональными требованиями системы; это делают тестеры программного обеспечения.
* **Нефункциональное тестирование.** Этот тип тестирования черного ящика связан не с тестированием конкретной функциональности, а с нефункциональными требованиями, такими как производительность, масштабируемость, удобство использования.
* **Регрессионное тестирование** — [Регрессионное тестирование](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/regressionnoe-testirovanie-2) проводится после того, как исправления кода, обновления или любое другое обслуживание системы для проверки того, что новый код не затронул существующий код.

## Инструменты, используемые для тестирования черного ящика:

Инструменты, используемые для тестирования черного ящика, во многом зависят от типа тестирования черного ящика, которое вы делаете.

* Для функциональных / регрессионных тестов вы можете использовать — [QTP](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-qtp/uchebnik-qtp) , [Selenium](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-selenium/uchebnik-selenium)
* Для [нефункциональных](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-jmeter/uchebnik-jmeter) тестов вы можете использовать — [LoadRunner](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-loadrunner/uchebnik-loadrunner) , [Jmeter](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-jmeter/uchebnik-jmeter)

## Второй метод - тестирование белого ящика

Проверка белого ящика - это подробное исследование внутренней логики и структуры кода. Тестирование с использованием белого ящика также называется **тестированием стекла** или **открытым тестированием**. Чтобы выполнить тестирование **белого ящика** в приложении, тестер должен знать внутреннюю работу кода.

Тестер должен заглянуть внутрь исходного кода и выяснить, какое устройство / блок кода ведет себя некорректно.

## Третий метод - тестирование серых ящиков

Тестирование на серой коробке - это метод тестирования приложения с ограниченным знанием внутренней работы приложения. При тестировании программного обеспечения фраза, чем больше вы знаете, тем лучше переносит массу при тестировании приложения.

Освоение домена системы всегда дает тестеру преимущество над кем-то с ограниченными знаниями домена. В отличие от тестирования черного ящика, где тестер тестирует только пользовательский интерфейс приложения; при тестировании в сером полете тестер имеет доступ к проектной документации и базе данных. Имея эти знания, тестер может подготовить лучшие тестовые данные и сценарии тестирования при составлении плана тестирования.

# 2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ УЧЁТ КОНТИНГЕНТА УЧАЩИХСЯ УЧЕБНОГО ЗАВЕДЕНИЯ

## 2.1. Разработка тестовой документации (тест-дизайн)

Документ описывает методы и подходы к тестированию, которые будут использоваться тестировщиками отдела тестирования, для тестирования приложения. План тестирования может использоваться как тестировщиками, так и менеджерами, разработчиками. Объект тестирования — это деятельность, направленная на проверку работоспособности функций приложения учёта контингента учащихся учебного заведения.

Целью тестирования приложения учёта контингента учащихся учебного заведения является проверка корректной работы.

Итогом процесса тестирования будут следующие материалы:

1. Определить существующую информацию о проекте и программных компонентах, подлежащих тестированию.
2. Описать стратегии тестирования, которые будут использоваться.
3. Определить необходимые ресурсы для проведения работ по тестированию.
4. Привести результаты тестирования.

Результаты будут отправлены заказчику в виде отчетов.

Условия для тестирования. Приложение должно удовлетворять потребность пользователя в активностях, связанных с просмотром информации о студентах.

Стратегия процесса тестирования. Приведенный ниже план тестирования является формальным, так как для построения развернутого плана необходимо понимание текущего состояния проекта.

Основными задачами тестирования являются:

* проведение функционального тестирования каждого модуля и компонента системы для обеспечения его соответствия функциональным требованиям;
* тестирование данных и целостности базы данных.

Виды тестирования. Для решения указанных выше задач тестирования будут использоваться следующие виды тестирования.

1. Тестирование данных и целостности базы данных

Базы данных должны тестироваться как отдельные системы внутри учёта контингента учащихся учебного заведения. Эти системы должны тестироваться отдельно от приложений (таких как интерфейс доступа к данным).

Необходимо провести дополнительное исследование СУБД на тему того, какие инструменты/техники существуют для выполнения нижеописанного тестирования.

1.1 Цель тестирования

Убедится в том, что методы доступа к данным работают правильно и без нарушения целостности БД.

1.2 Способы

* Вызвать каждый метод доступа к БД, предоставляя правильные и не правильные данные (или запросы к данным).
* Исследовать БД на предмет корректного заполнения ее данными, корректной обработки событий

1.3 Критерий завершенности

Все методы и процедуры БД функционируют так, как им положено и без нарушения целостности самой БД.

1.4 Особые замечания

* При тестировании может понадобиться среда разработки СУБД или драйвера для корректного подключения к базам данных.
* Процедуры должны вызываться вручную.
* Для повышения видимости неприемлемых событий БД необходимо использовать небольшие БД или БД с ограниченным количеством записей.

2. Функциональное тестирование

1.1 Цель тестирования

Функциональное тестирование состоит в том, чтобы убедиться, что весь программный продукт работает в соответствии с требованиями, и в приложении не появляется существенных ошибок.

1.2 Способы

**Авторизация**

* Авторизация пользователя

**Просмотр каталога**

* Просмотр списка блюд
* Добавление/Удаление блюд корзины
* Добавление/Удаление блюд на склад
* Выход

1.3 Критерий завершенности

Программный продукт должен пройти все запланированные тесты.

1.4 Особые замечания

Ожидаемые результаты возникают при использовании достоверных данных.

Соответствующие сообщения об ошибках или предупреждения отображаются, когда используются неверные данные.

Подготовлено тестовое окружение, приложение готово к тестированию на тестовой площадке.

Не будет проведено нагрузочное и тестирование безопасности в виду отсутствия необходимых ресурсов.

Отчеты об ошибках создаются для того, чтобы предоставить команде разработчиков и руководителю проекта исчерпывающую информацию об обнаруженных ошибках. Они должны быть полезны при определении причин ошибок и их исправлении.

Продукт должен работать в соответствии с требованиями и техническим заданием. Продукт не должен содержать критических и блокирующих дефектов в окончательной версии проекта.

## 2.2. Разработка тестовых сценариев

После установки базы данных можно приступить к ее использованию в тестах. В большинстве простых случаев в наборе тестов есть отдельная база данных, которая совместно используется несколькими тестами в нескольких классах тестирования, поэтому нам нужна некоторая логика, чтобы убедиться, что база данных создана и заполняется ровно один раз в течение всего времени существования тестового запуска.

При использовании Xunit это можно сделать с помощью [средства](https://xunit.net/docs/shared-context#class-fixture) класса, представляющего базу данных и совместно используемой для нескольких тестовых запусков:

public class TestDatabaseFixture

{

private const string ConnectionString = @"Server=(localdb)\mssqllocaldb;Database=EFTestSample;Trusted\_Connection=True";

private static readonly object \_lock = new();

private static bool \_databaseInitialized;

public TestDatabaseFixture()

{

lock (\_lock)

{

if (!\_databaseInitialized)

{

using (var context = CreateContext())

{

context.Database.EnsureDeleted();

context.Database.EnsureCreated();

context.AddRange(

new Blog { Name = "Blog1", Url = "http://blog1.com" },

new Blog { Name = "Blog2", Url = "http://blog2.com" });

context.SaveChanges();

}

\_databaseInitialized = true;

}

}

}

public BloggingContext CreateContext()

=> new BloggingContext(

new DbContextOptionsBuilder<BloggingContext>()

.UseSqlServer(ConnectionString)

.Options);

}

При создании экземпляра приведенного выше средства используется [EnsureDeleted()](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.storage.idatabasecreator.ensuredeleted#microsoft-entityframeworkcore-storage-idatabasecreator-ensuredeleted) для удаления базы данных (если она существует из предыдущего запуска), а затем [EnsureCreated()](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.storage.idatabasecreator.ensurecreated#microsoft-entityframeworkcore-storage-idatabasecreator-ensurecreated) для ее создания с помощью последней конфигурации модели ([см. документацию по этим API](https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/ensure-created)). После создания базы данных средство заменит его некоторыми данными, которые могут использовать наши тесты. Стоит потратить некоторое время на размышления о начальных данных, так как изменение его позже для нового теста может привести к сбою существующих тестов.

Чтобы использовать приспособление в тестовом классе, просто реализуйте IClassFixture его над типом светильника, а xUnit внедряет его в конструктор:

public class BloggingControllerTest : IClassFixture<TestDatabaseFixture>

{

public BloggingControllerTest(TestDatabaseFixture fixture)

=> Fixture = fixture;

public TestDatabaseFixture Fixture { get; }

Теперь класс тестирования имеет Fixture свойство, которое можно использовать тестами для создания полнофункциональный экземпляр контекста:

[Fact]

public void GetBlog()

{

using var context = Fixture.CreateContext();

var controller = new BloggingController(context);

var blog = controller.GetBlog("Blog2").Value;

Assert.Equal("http://blog2.com", blog.Url);

}

Вот здесь можно заметить некоторые блокировки в логике создания светильника выше. Если приспособление используется только в одном тестовом классе, оно гарантированно будет создано ровно один раз с помощью xUnit; Но обычно один и тот же компонент базы данных используется в нескольких тестовых классах. xUnit предоставляет [средства сбора](https://xunit.net/docs/shared-context#collection-fixture), но этот механизм предотвращает параллельное выполнение тестовых классов, что важно для производительности тестирования. Чтобы безопасно управлять этим с помощью средства xUnit класса, берем простую блокировку при создании и заполнении базы данных и используем статический флаг, чтобы мы никогда не делали это дважды.

В приведенном выше примере показан тест только для чтения, который является простым случаем с точки зрения изоляции теста: так как ничего не изменяется, вмешательство теста невозможно. В отличие от этого, тесты, которые изменяют данные, являются более проблематичными, так как они могут вмешиваться друг в друга. Одним из распространенных способов изоляции тестов является перенос теста в транзакцию и откат этой транзакции в конце теста. Так как в базе данных ничего не зафиксировано, другие тесты не видят никаких изменений и интерференции не будут избегать.

Ниже приведен метод контроллера, который добавляет блог в нашу базу данных:

[HttpPost]

public ActionResult AddBlog(string name, string url)

{

\_context.Blogs.Add(new Blog { Name = name, Url = url });

\_context.SaveChanges();

return Ok();

}

Мы можем протестировать этот метод следующим образом:

[Fact]

public void AddBlog()

{

using var context = Fixture.CreateContext();

context.Database.BeginTransaction();

var controller = new BloggingController(context);

controller.AddBlog("Blog3", "http://blog3.com");

context.ChangeTracker.Clear();

var blog = context.Blogs.Single(b => b.Name == "Blog3");

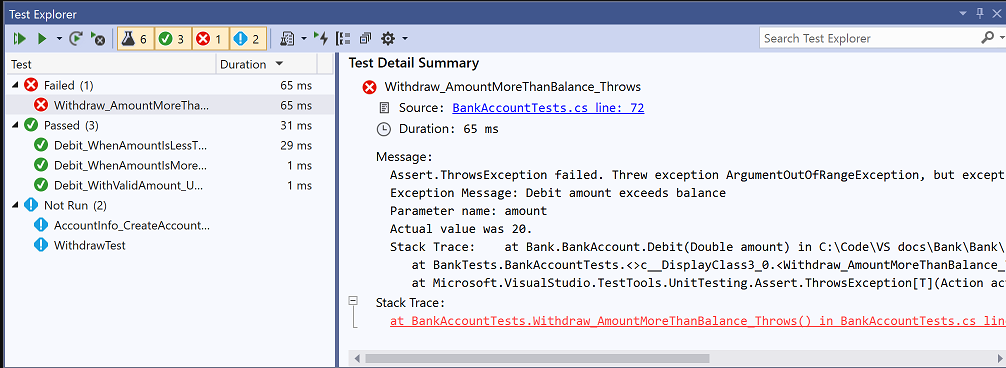
Assert.Equal("http://blog3.com", blog.Url);

}

Некоторые примечания к приведенному выше коду теста:

* Запускаем транзакцию, чтобы убедиться, что приведенные ниже изменения не зафиксированы в базе данных и не вмешиваются в другие тесты. Так как транзакция никогда не фиксируется, она неявно откатывается в конце теста при удалении экземпляра контекста.
* После внесения нужных обновлений очистим средство отслеживания [ChangeTracker.Clear](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.changetracking.changetracker.clear)изменений экземпляра контекста, чтобы убедиться, что мы фактически загружаем блог из базы данных ниже. Вместо этого можно использовать два экземпляра контекста, но затем необходимо убедиться, что одна и та же транзакция используется обоими экземплярами.
* Возможно, даже захотите запустить транзакцию в приспособлении CreateContext, чтобы тесты получали экземпляр контекста, который уже находится в транзакции, и готов к обновлению. Это может помочь предотвратить случаи, когда транзакция случайно забыта, что приводит к проверке помех, которые могут быть трудно отлаживать. Кроме того, может потребоваться разделить тесты только для чтения и написать их в различных классах тестирования.

При построении проекта тестирования тесты появляются в обозревателе тестов. Если обозреватель тестов не виден, выберите Тест в меню Visual Studio, Windows, затем обозреватель тестов (или нажмите клавиши CTRL + E, T).



Обозреватель модульных тестов

При выполнении, написании и повторном запуске тестов обозреватель тестов может отображать результаты в группах Неудачные тесты, Пройденные тесты, Пропущенные тесты и Незапущенные тесты. Можно выбирать различные группы по параметрам на панели инструментов.

Кроме того, можно фильтровать тесты по совпадению текста в поле поиска на глобальном уровне или с помощью одного из предустановленных фильтров. Можно запустить любую выборку тестов в любое время. Результаты запущенного теста появляются сразу же в строке "успешно/не успешно" наверху окна обозревателя. Детальная информация результата метода тестирования отображается при выборе теста.

# ЗАКЛЮЧЕНИЕ

Сегодня тестирование является неотъемлемой частью процесса производства программных продуктов. Качественное тестирование помогает своевременно выявлять и исправлять ошибки, тем самым уменьшая риски и затраты на разработку программного обеспечения. При автоматизации тестирования скорость и качество тестирования может повыситься, что приведет к еще большему снижению издержек и повышению качества.

В данной работе была проанализирована эффективность внедрения ИС «Доставка еды (на двоих)» Для достижения поставленной цели были решены такие теоретические задачи, как описание теоретических основ тестирования, классификация и описание различных видов тестирования, описание методологий, анализ процесса тестирования, выявление и описание критериев корректного построения процесса тестирования. В работе также были определены критерии эффективности процесса тестирования, описаны и проанализированы формулы, позволяющие выразить эффективность данного процесса в денежном эквиваленте.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 34.601 – 90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.
2. ГОСТ 34.602 - 2020. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.
3. ГОСТ 19.201 - 78 ЕСПД. Техническое задание. Требования к содержанию и оформлению.
4. ГОСТ 19.202 - 78 ЕСПД. Спецификация. Требования к содержанию и оформлению.
5. ГОСТ Р ИСО/МЭК 12207. Процессы жизненного цикла программных средств.
6. Золотов С.Ю. Проектирование информационных систем: Учебно-методическое пособие. Методические рекомендации для выполнения курсового проекта, лабораторных работ и практических занятий по дисциплине «Проектирование информационных систем» - Томск: ТУСУР, 2013. - 34 с.
7. Шнайдер, Роберт Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных; М.: Лори, 2010. - 361 c
8. Петкович, Душан Microsoft SQL Server 2012. Руководство для начинающих / Душан Петкович. - М.: БХВ-Петербург, 2012. - 460 c.
9. Тейлор, Аллен SQL для чайников / Аллен Тейлор. - М.: Вильямс, 2014. - 416 c.
10. Браст, Э.Дж. Разработка приложений на основе Microsoft SQL Server 2008 / Э.Дж. Браст. - М.: Русская Редакция, 2010. - 751 c.
11. Хетагуров, Я. А. Проектирование автоматизированных систем обработки информации и управления (АСОИУ). Учебник / Я.А. Хетагуров. - М.: Бином. Лаборатория знаний, 2015. - 240 c.
12. Кристофер, Д. Маннинг Введение в информационный поиск / Кристофер Д. Маннинг, ПрабхакарРагхаван ,ХайнрихШютце. - М.: Вильямс, 2014. - 528 c.
13. Бишоп Дж. С# в кратком изложении; Бином. Лаборатория знаний - М., 2015. - 234 c.
14. ГриффитсИэн Программирование на C# 5.0; Эксмо - М., 2014. - 580 c.
15. Гуриков С. Р. Введение в программирование на языке Visual C#; ИЛ - Москва, 2013. - 448 c.
16. Шилдт Герберт C# 4.0. Полное руководство; Вильямс - М., 2015. - 291 c.
17. Эндрю Троелсен Язык программирования C# 5.0 и платформа .NET 4.5; Диалектика / Вильямс - М., 2015. - 126 c.